Cloud Security Mechanisms
Homomorphic Encryption

Hubert Hesse, Christoph Matthies

What is that?

```
operation(plain)
```

What is that?

```
operation(plain)
```

==

```
decrypt(operation'(encrypt(plain)))
```

What is that?

```
operation(plain)
```

==

```
decrypt(operation'(encrypt(plain)))
```

i.e. outputs of operations on encrypted data are still usable

Current context

July 2013:
Change in "De-Mail-Gesetz" defining De-Mail as secure [1]

- Needs to be decrypted by provider to "check for viruses"
- (Secret) key on server of provider
  - Server becomes juicy target
- Homomorphic encryption
  - Can check without decryption

[1] http://www.spiegel.de/netzwelt/netzpolitik/de-mail-bundestag-erklaert-bundes-mail-per-gesetz-als-sicher-a-895361.html

Use cases

- Medical records
  - Analyze disease / treatment without disclosing them
  - Search for DNA markers without revealing DNA
  - "Digitale Krankenakte"

- Spam filtering
  - Blacklisting encrypted mails
  - Third parties can scan your PGP traffic

Doing something without knowing what

Homomorphism

groups $(P, \oplus)$ and $(C, \otimes)$

relation $f : P \longrightarrow C$

f is a **group homomorphism** in P and C, if:

$\forall\ a,b \in P: \quad f(a \oplus b) = f(a) \otimes f(b)$

Especially:

$\forall\ a,b \in P: \quad a \oplus b = f^{-1}(\, f(a) \otimes f(b)\,)$

Examples

groups $(R, +)$ and $(R^*, \times)$

function: $R \rightarrow R$

$$\exp(x+y) = \exp(x) \times \exp(y)$$

$$10^{x+y} = 10^x \times 10^y$$

$$\ln(a \times b) = \ln(a) + \ln(b)$$

be aware, mapping from one operation to another

Practical example

In **RSA**,
multiplication is
(accidentally)
a homomorphism

Imagine

width = 7
height = 3

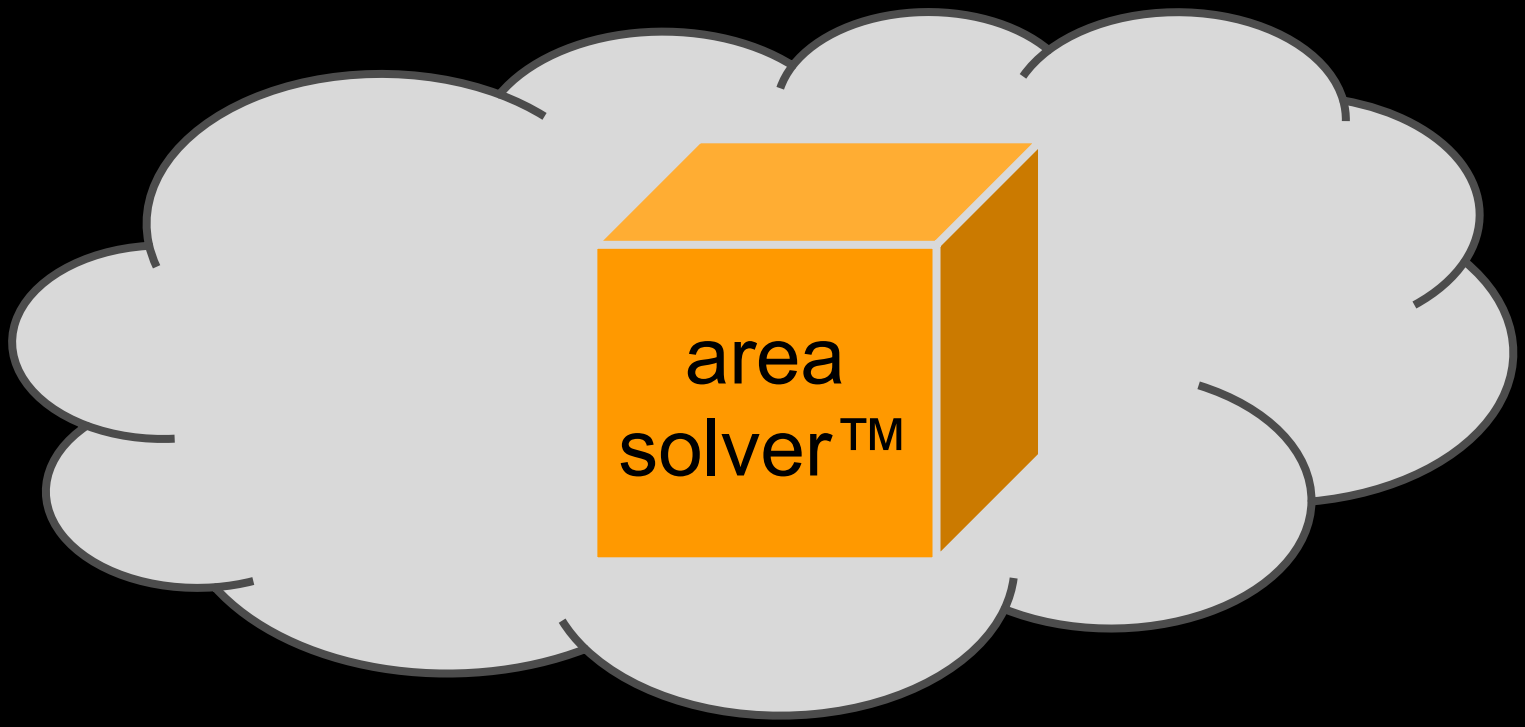
Imagine

what's the area?

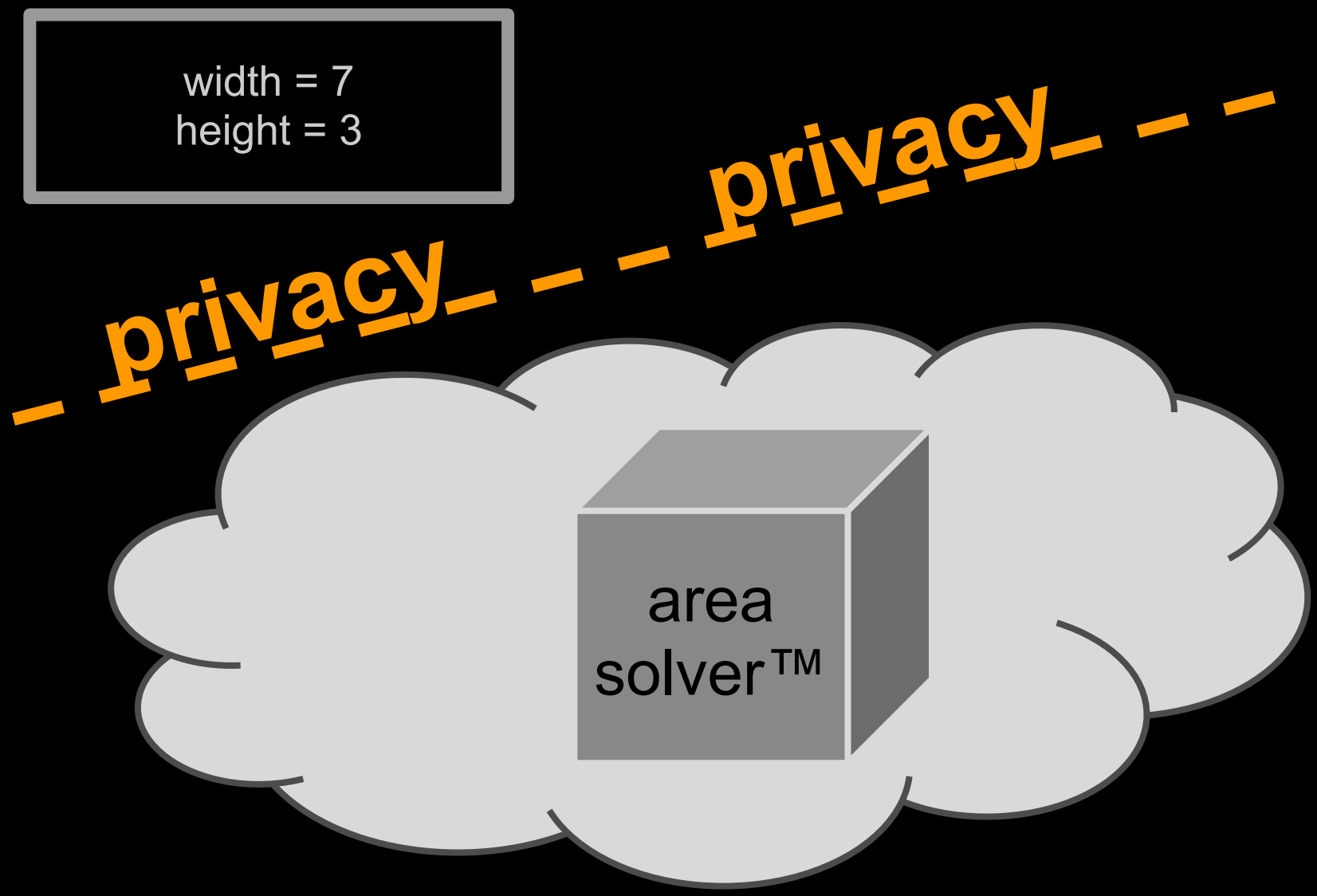
width = 7
height = 3

Enter the cloud

width = 7
height = 3

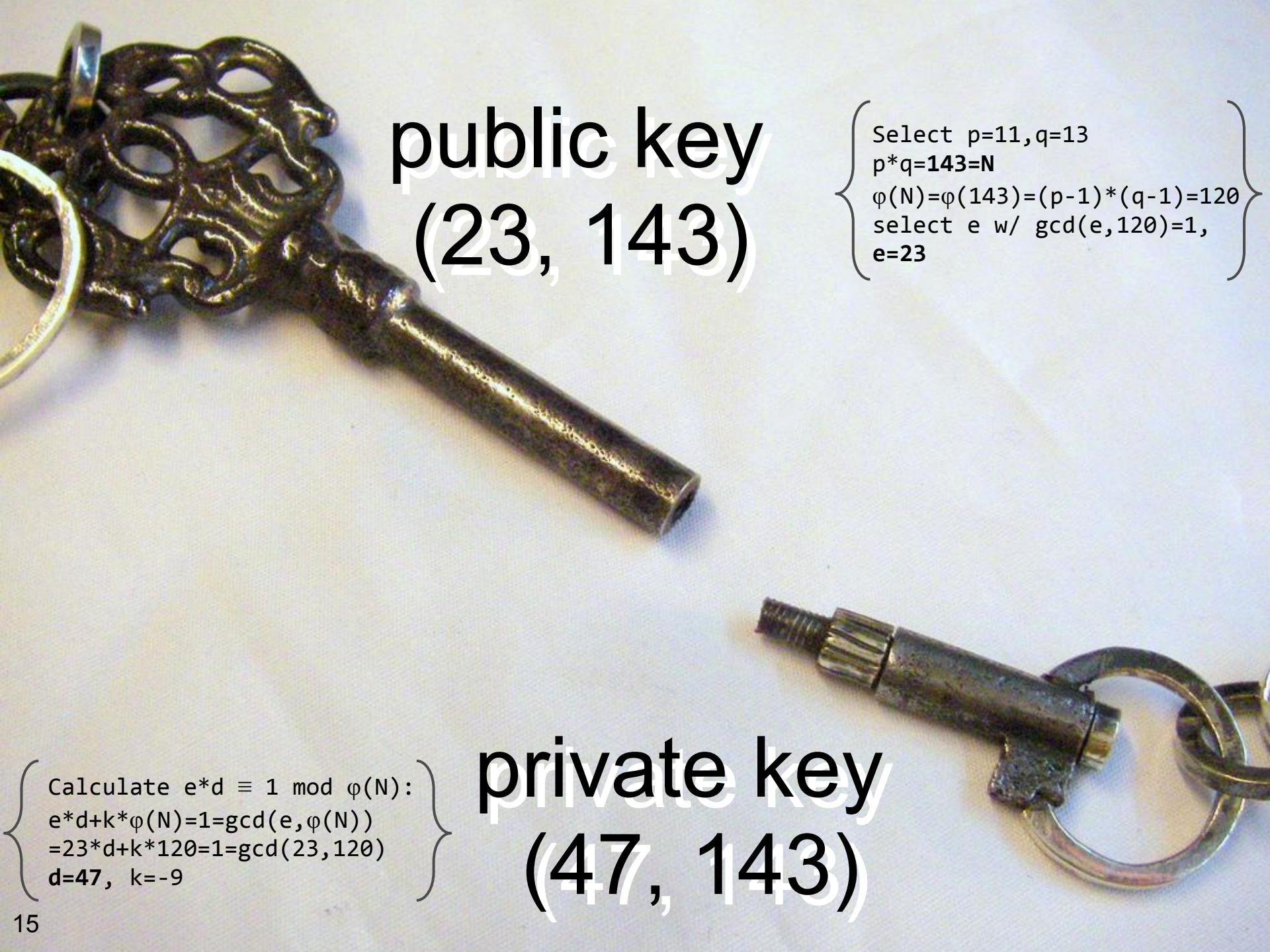


Enter the cloud



Enter the cloud

RSA to the rescue



public key (23, 143)

Select p=11,q=13
p*q=**143=N**
φ(N)=φ(143)=(p-1)*(q-1)=120
select e w/ gcd(e,120)=1,
e=23

private key (47, 143)

Calculate e*d ≡ 1 mod φ(N):
e*d+k*φ(N)=1=gcd(e,φ(N))
=23*d+k*120=1=gcd(23,120)
d=47, k=-9

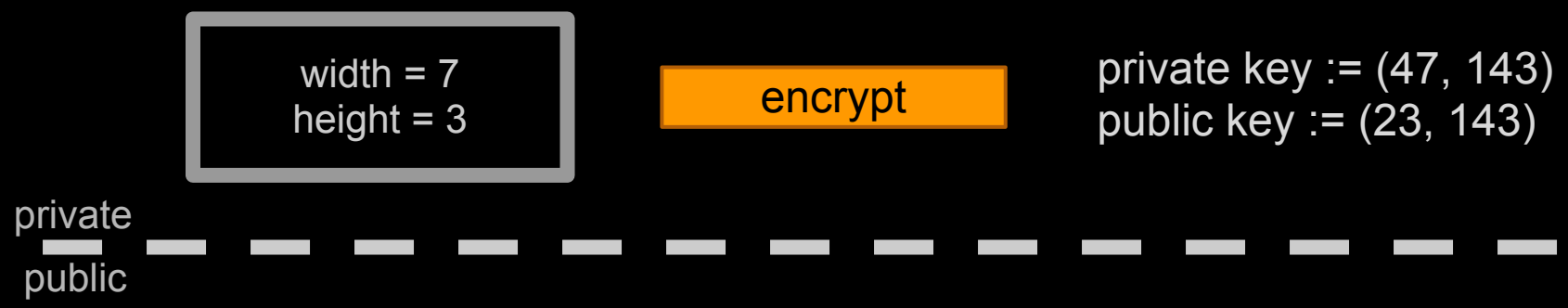
wait, RSA?

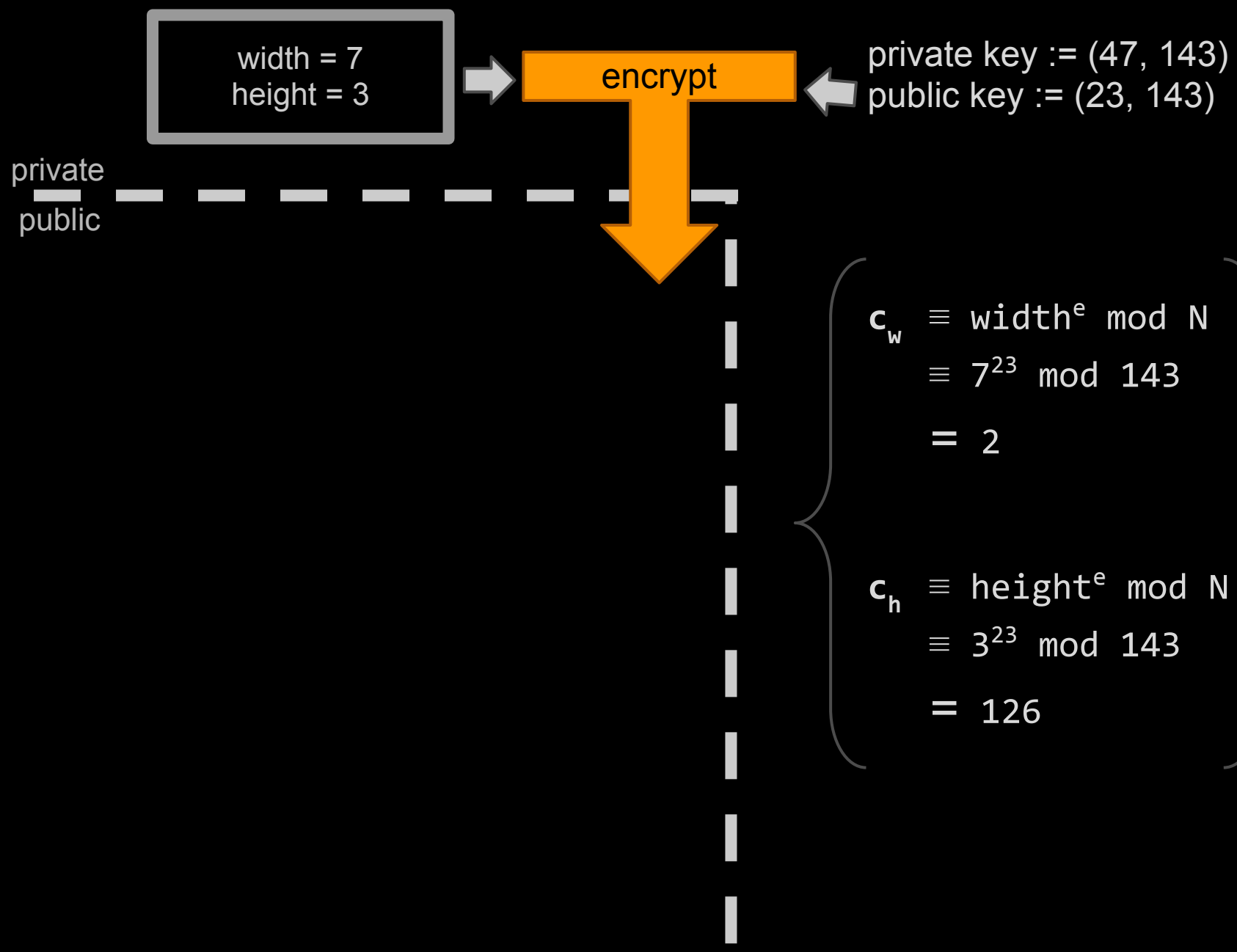
Encryption in RSA

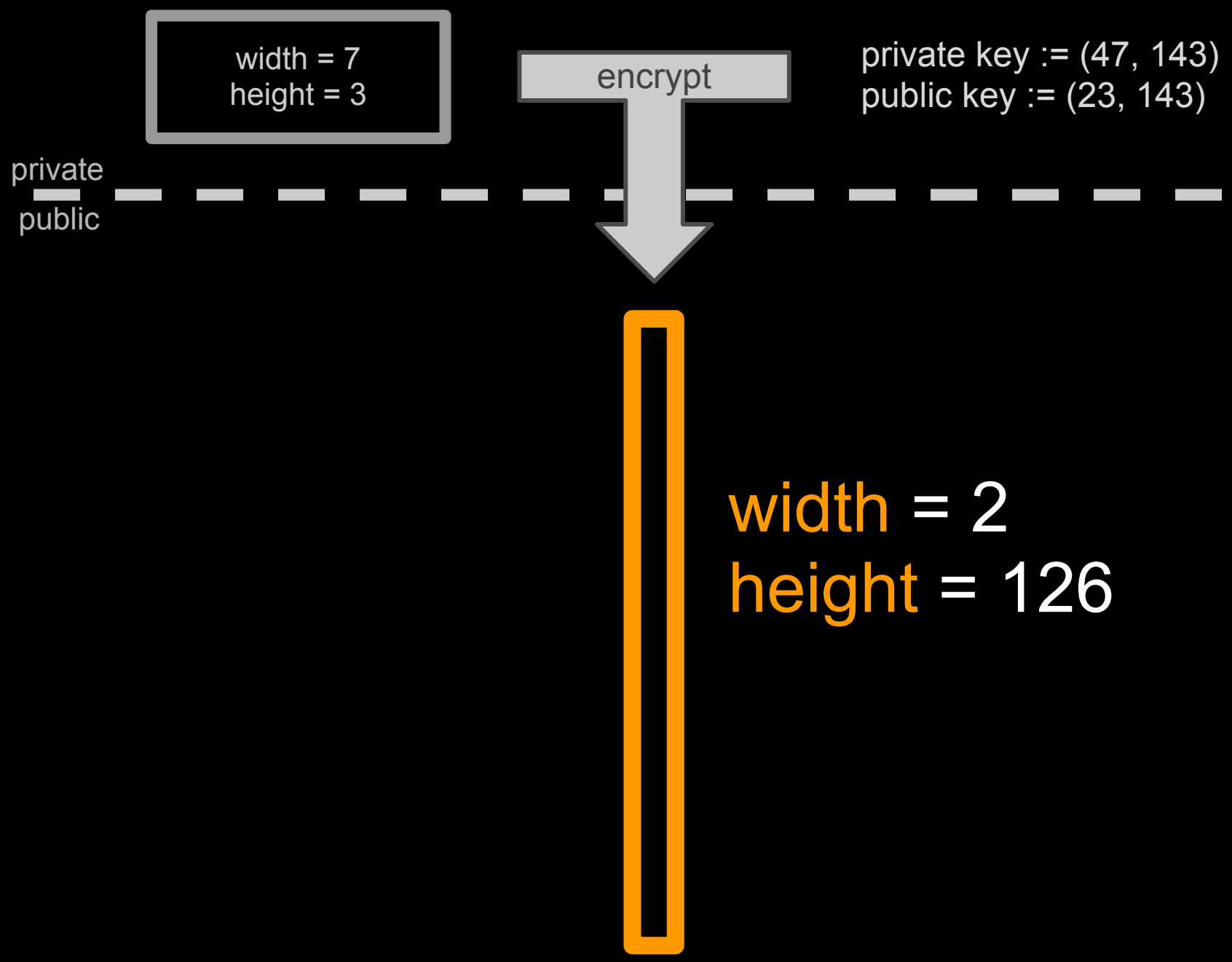
$$\text{encrypt(message)} \equiv \text{message}^{\text{pub. key exponent}} \bmod \text{RSA-modulus}$$

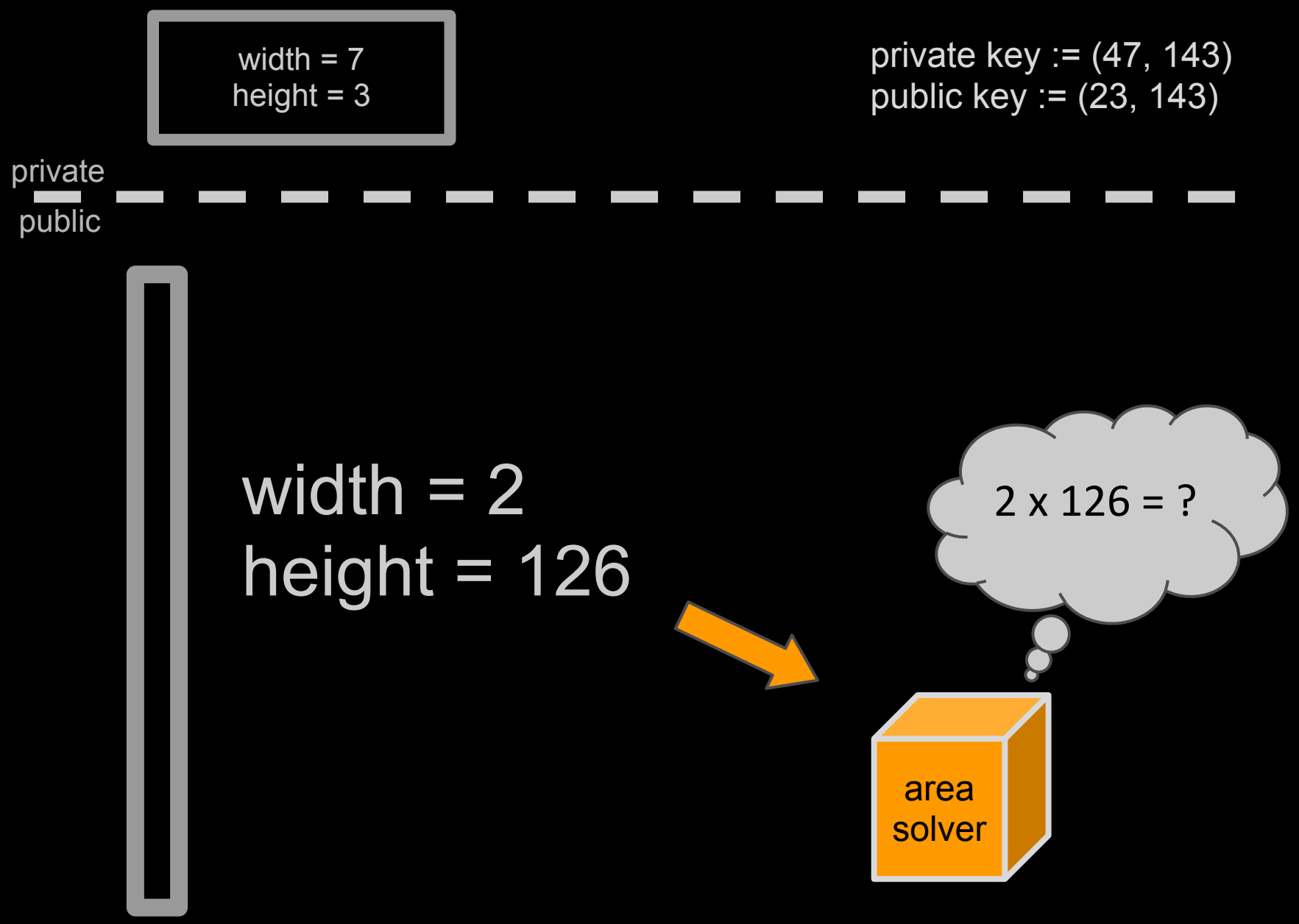
Homomorphic property

$$\text{encrypt}(m) \times \text{encrypt}(n) = \text{encrypt}(m \times n)$$


width = 7
height = 3
encrypt
private key := (47, 143)
public key := (23, 143)
private
public


width = 7
height = 3
encrypt
private key := (47, 143)
public key := (23, 143)
private
public
$c_w \equiv width^e \bmod N$
$\equiv 7^{23} \bmod 143$
$= 2$
$c_h \equiv height^e \bmod N$
$\equiv 3^{23} \bmod 143$
$= 126$


width = 7
height = 3
encrypt
private key := (47, 143)
public key := (23, 143)
private
public
width = 2
height = 126


width = 7
height = 3
private key := (47, 143)
public key := (23, 143)
private
public
width = 2
height = 126
2 x 126 = ?
area
solver

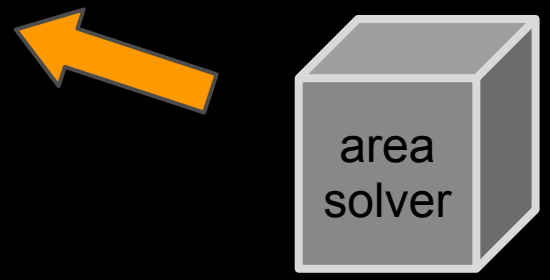
width = 7
height = 3

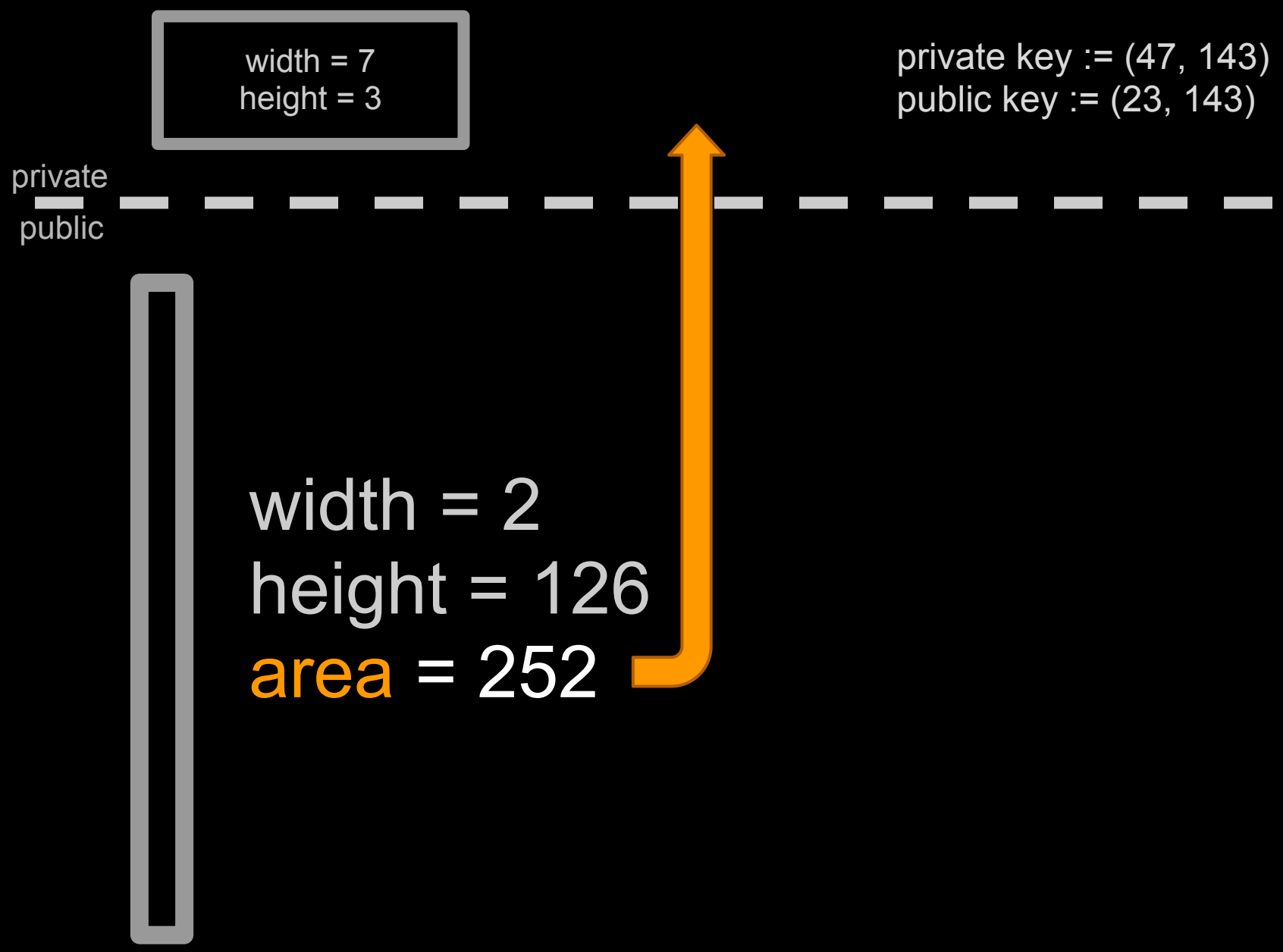
private key := (47, 143)
public key := (23, 143)

private

public

width = 2
height = 126
area = 252




width = 7
height = 3
private key := (47, 143)
public key := (23, 143)
private
public
width = 2
height = 126
area = 252

area = 21

width = 7
height = 3

decrypt

private key := (47, 143)
public key := (23, 143)

private

public

width = 2
height = 126
area = 252

area = decrypt(252)
= 21

$$\text{area} \equiv \text{cipher}^d \bmod N$$
$$\equiv 252^{47} \bmod 143$$
$$= 21$$

area = 21

width = 7
height = 3

decrypt

private key := (47, 143)
public key := (23, 143)

private

public

width = 2
height = 126
area = 252

area = decrypt(252)
= 21
= 7 x 3

(sanity check)

Different homomorphisms

- RSA allows only multiplication
  - Other operations on ciphertext (e.g. +) break decryption

circumference calculation

```
correct:    3*2 + 7*2 = 20
encrypted:  2*2 + 2*126 = 256
decryption: 256^47 mod 143 = 42
            42 ≠ 20 □
```

- Other schemes allow different operations (e.g. + and -)

- Algebra homomorphisms allows x and +
  - **Much** more powerful

f: A→B alg. hom. ⇔ ∀k∈K; x,y∈A:

- $f(k*x)=k*f(x)$
- $f(x+y)=f(x)+f(y)$
- $f(x*y)=f(x)*f(y)$

Different homomorphisms

- RSA allows only multiplication
  - Other operations on ciphertext (e.g. +) break decryption

circumference calculation

```
correct:    3*2 + 7*2 = 20
encrypted:  2*2 + 2*126 = 256
decryption: 256^47 mod 143 = 42
            42 ≠ 20 □
```

- Other schemes allow different operations (e.g. + and -)

- Algebra homomorphisms allows x and +
  - **Much** more powerful

f: A→B alg. hom. ⇔ ∀k∈K; x,y∈A:

- $f(k*x)=k*f(x)$
- $f(x+y)=f(x)+f(y)$
- $f(x*y)=f(x)*f(y)$

Need to select appropriate homomorphic encryption scheme for application

System	Plaintext operation	Cipher operation
RSA	×	×
Paillier	+, − m×k, m+k	×, ÷ c^k, $c \times g^k$
ElGamal	× m×k, m^k	× c×k, c^k
Goldwasser-Micali	⊕	×
Benaloh	+, −	×, ÷
Naccache-Stern	+, − m×k	×, ÷ c^k
Sander-Young-Yung	×	+
Okamoto-Uchiyama	+, − m×k, m+k	×, ÷ c^k, c+e(k)
Boneh-Goh-Nissim	Paillier (+, −, m×k, m+k) × (once)	Paillier bilinear pairing
US 7'995'750 / Rot13	+	+

Pollution

(simplified)

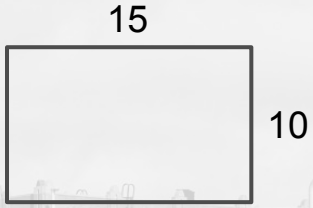
- Operations on ciphertext accumulate "noise"
  - Addition adds noise, multiplication multiplies it
  - Noise gets too high ⟶ decryption fails
- These "limited" algebra homomorphism schemes

Somewhat Homomorphic Encryption Scheme

Pollution Example

- Using small N in RSA and large inputs
  - When output larger than RSA-modulus, decryption fails

Calculate area of square using RSA



10*15=150

Encryption:

$c\_w \equiv 10^{23} \bmod 143$

$\equiv 43$

$c\_h = 15^{23} \bmod 143$

$= 20$

$c\_a = 43*20 = 860$

Decryption:

$a \equiv 860^{47} \bmod 143$

$\equiv 150 \bmod 143$

$\equiv 7$

$\mathbf{7 \neq 150}$ ☐

Beyond + and ×

Every* program can be expressed in terms of a digital circuit.

* referentially transparent, ie. w/o side effects, today() is not ref. transparent

Beyond + and ×

Every digital circuit can be expressed in terms of AND, OR, and NOT.

Beyond + and ×

Every digital circuit can be expressed in terms of AND, OR, and NOT.

(remember Disjunctive Normal Forms?)

Beyond + and ×

Every digital circuit can be expressed in terms of AND and XOR.

XOR(x, 1) = NOT(x)

NOT(AND(NOT(x), NOT(y))) = !(!x & !y) = OR(x, y)

Fully homomorphic encryption

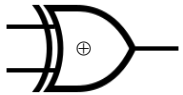
With $\wedge$ and $\oplus$ we can represent *any* operation

Circuit Encryption

- Assume homomorphic enc:
  - 0-bits ⟶ even ints
  - 1-bits ⟶ odd ints

(+ random r * secret p mod p)

  - $\oplus \rightarrow +$



{ simple truth tables }

  - $\wedge \rightarrow \times$

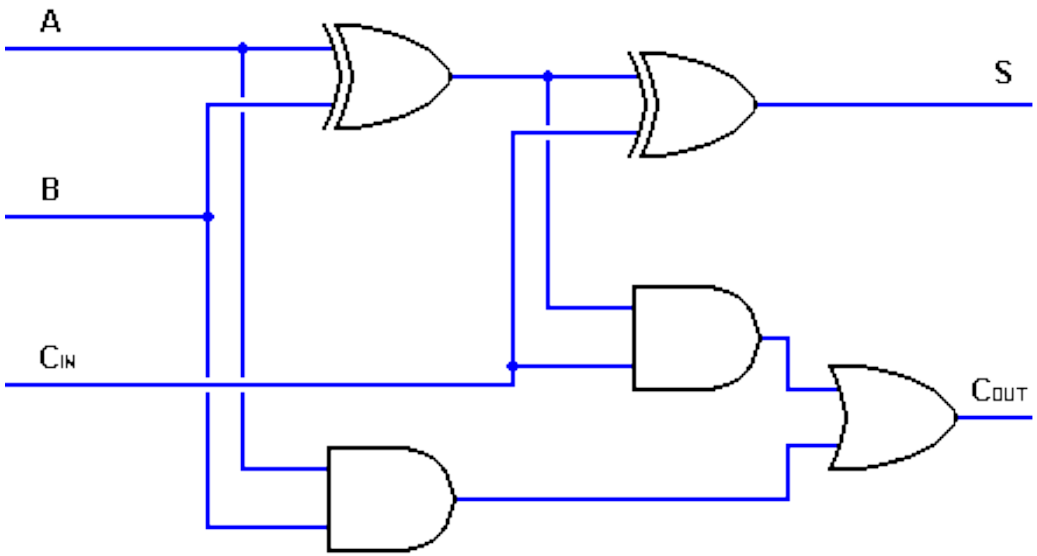
  - Define: $\circ$ = (a + b) + (a x b) (Logical OR)

{ OR = $(a \wedge b) \wedge (a \oplus b)$ }

Represent and evaluate any circuit in hom. space!

Circuit Encryption
Toy example

- Single Bit Adder
  - A,B: inputs, C_{in}: carry-in, S: sum, C_{out}: carry-out



$$S = ((A \oplus B) \oplus C)$$

$$C_{out} = (C_{in} \wedge B) \vee ((A \oplus B) \wedge C_{in})$$

Circuit Encryption

Toy example

$$S = ((A \oplus B) \oplus C)$$

$$C_{out} = (C_{in} \wedge B) \vee ((A \oplus B) \wedge C_{in})$$

$$S = ((A + B) + C)$$

$$C_{out} = (C_{in} \times B) \circ ((A+B) \times C_{in})$$

map operators

Toy example - calc. S

S = ((A + B) + C)

apply

encrypted

A	B	C_{in}	S	C_{out}
1	0	1	0	1
3	4	7	?	?

S = ((3 + 4) + 7) = ?

Toy example - calc. S

S = ((A + B) + C)

	A	B	C_{in}	S	C_{out}
	1	0	1	0	1
encrypted	3	4	7	14	?

S = ((3 + 4) + 7) = 14 ≙ 0

Toy example - calc. C_{out}

$$C_{out} = (C_{in} \times B) \circ ((A + B) \times C_{in})$$

apply

$$C_{out} = (3 \times 4) \circ ((3 + 4) \times 7)$$

$$= 12 \circ 49$$

$$= (12 + 49) + (12 * 49)$$

$$= 61 + 588 = 649 \mathrel{\hat{=}} 1$$

$\circ = (a + b) + (a \times b)$

A	B	C_{in}	S	C_{out}
1	0	1	0	1
3	4	7	14	649

Circuit Encryption

- Assume homomorphic enc:
  - 0-bits ⟶ even ints
  - 1-bits ⟶ odd ints

(actually mod a secret p)

  - $\oplus \longrightarrow +$
  - $\wedge \longrightarrow \times$

{ simple truth tables }

  - Define: $\circ = (a + b) + (a \times b)$ (Logical OR)

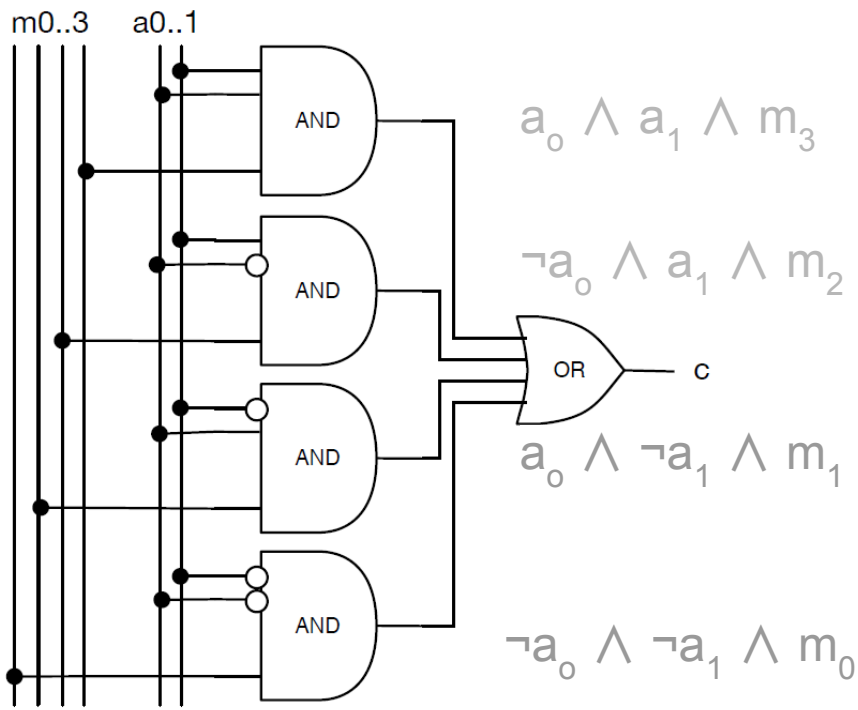
{ OR = (a ∧ b) ∧ (a ⊕ b) }

Represent and evaluate any circuit in hom. space!

But memory access patterns could be leaked

Circuit Enc. Example

- Encrypted Memory Access
- We have our memory $m_{0..3}$ (4bit) and want to reassign memory adresses $a_0 a_1$

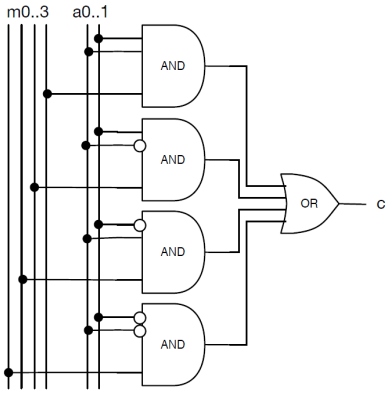


m_0	m_1	m_2	m_3	a_0	a_1
1	x	x	x	0	0
x	1	x	x	1	0
x	x	1	x	0	1
x	x	x	1	1	1

Extend to required amount memory & address lines

Encrypted Memory Access

- Map to integer arithmetic[1]



$$row_3 = a_o \wedge a_1 \wedge m_3$$

$$row_2 = \neg a_o \wedge a_1 \wedge m_2$$

$$row_1 = a_o \wedge \neg a_1 \wedge m_1$$

$$row_0 = \neg a_o \wedge \neg a_1 \wedge m_0$$

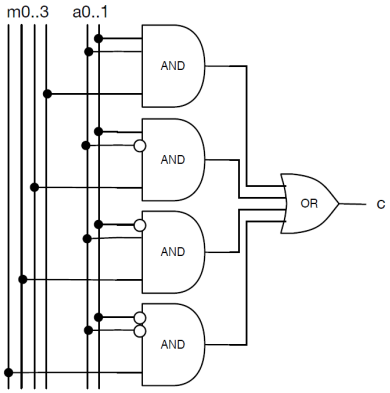
Calculate circuit in logical space

$$c = row_0 \vee row_1 \vee row_2 \vee row_3$$

[1] M Brenner, J Wiebelitz, G von Voigt. Secret program execution in the cloud applying homomorphic encryption. Digital Ecosystems and …, 2011

- Map to integer arithmetic

m = {1, 0, 1, 0} a = 01



$$row_3 = a_o \wedge a_1 \wedge m_3$$

$$row_2 = \neg a_o \wedge a_1 \wedge m_2$$

$$row_1 = a_o \wedge \neg a_1 \wedge m_1$$

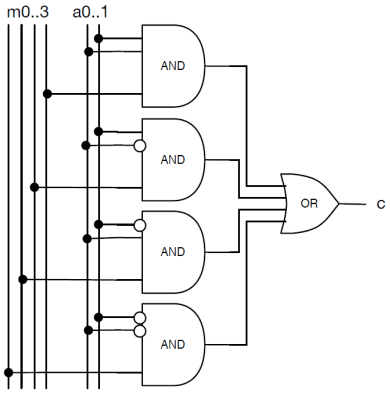
$$row_0 = \neg a_o \wedge \neg a_1 \wedge m_0$$

Calculate circuit in logical space

$$c = row_0 \vee row_1 \vee row_2 \vee row_3$$

- Map to integer arithmetic

m = {1, 0, 1, 0} a = 01



$$row_3 = a_o \wedge a_1 \wedge m_3$$

$$row_2 = \neg a_o \wedge a_1 \wedge m_2$$

$$row_1 = a_o \wedge \neg a_1 \wedge m_1$$

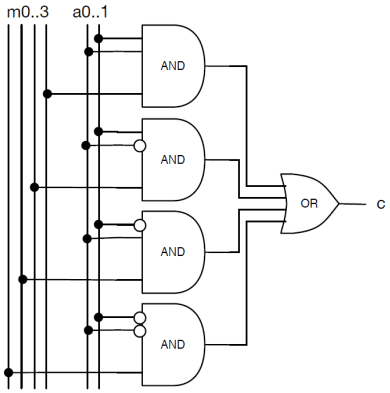
$$row_0 = \neg a_o \wedge \neg a_1 \wedge m_0$$

Calculate circuit in logical space

$$c = row_0 \vee row_1 \vee row_2 \vee row_3$$

- Map to integer arithmetic

m = {1, 0, 1, 0} a = 01



$$row_3 = a_o \wedge a_1 \wedge 0$$

$$row_2 = \neg a_o \wedge a_1 \wedge 1$$

$$row_1 = a_o \wedge \neg a_1 \wedge 0$$

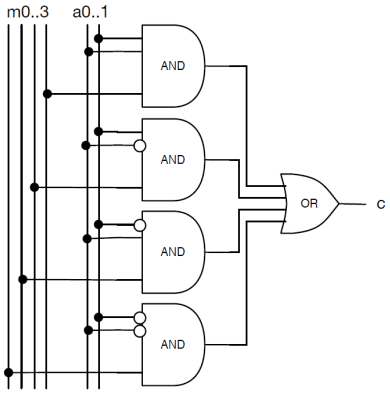
$$row_0 = \neg a_o \wedge \neg a_1 \wedge 1$$

Calculate circuit in logical space

$$c = row_0 \vee row_1 \vee row_2 \vee row_3$$

- Map to integer arithmetic

m = {1, 0, 1, 0} a = 01



$$row_3 = a_o \wedge a_1 \wedge 0$$

$$row_2 = \neg a_o \wedge a_1 \wedge 1$$

$$row_1 = a_o \wedge \neg a_1 \wedge 0$$

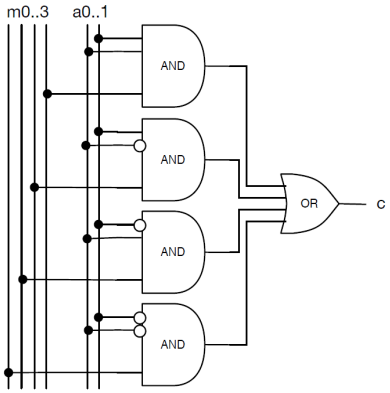
$$row_0 = \neg a_o \wedge \neg a_1 \wedge 1$$

Calculate circuit in logical space

$$c = row_0 \vee row_1 \vee row_2 \vee row_3$$

- Map to integer arithmetic

m = {1, 0, 1, 0} a = 01



$row_3 = 0 \wedge 1 \wedge 0$

$row_2 = 1 \wedge 1 \wedge 1$

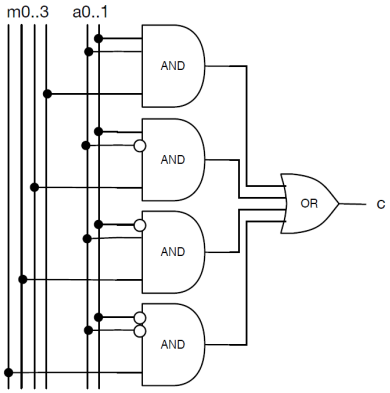
$row_1 = 0 \wedge 0 \wedge 0$

$row_0 = 1 \wedge 0 \wedge 1$

Calculate circuit in logical space

$$c = row_0 \vee row_1 \vee row_2 \vee row_3$$

- Map to integer arithmetic



m = {1, 0, 1, 0} a = 01

$row_3 = 0 \wedge 1 \wedge 0 = 0$

$row_2 = 1 \wedge 1 \wedge 1 = 1$

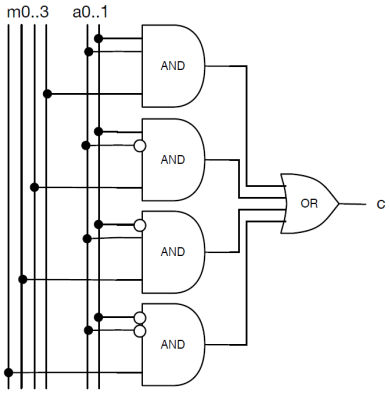
$row_1 = 0 \wedge 0 \wedge 0 = 0$

$row_0 = 1 \wedge 0 \wedge 1 = 0$

Calculate Circut in homorphic space

$$c = row_0 \vee row_1 \vee row_2 \vee row_3$$

- Map to integer arithmetic



m = {1, 0, 1, 0} a = 01

$row_3 = 0 \wedge 1 \wedge 0 = 0$

$row_2 = 1 \wedge 1 \wedge 1 = 1$

$row_1 = 0 \wedge 0 \wedge 0 = 0$

$row_0 = 1 \wedge 0 \wedge 1 = 0$

Calculate circuit in logical space

$c = 0 \vee 0 \vee 1 \vee 0 = 1$

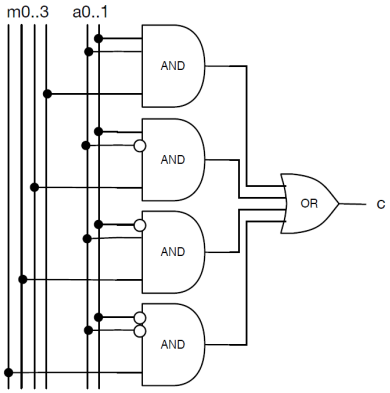
- Map to integer arithmetic

0-bits → even ints

1-bits → odd ints

$\bullet \rightarrow +$

$\wedge \rightarrow \times$



m = {1, 0, 1, 0} a = 01 m = {5, 4, 9, 6} a = {8, 3}

$row_3 = 0 \wedge 1 \wedge 0 = 0 \longrightarrow row_3 = (a_o \times a_1 \times 6)$

$row_2 = 1 \wedge 1 \wedge 1 = 1 \longrightarrow row_2 = (a_o + 1) \times a_1 \times 9$

$row_1 = 0 \wedge 0 \wedge 0 = 0 \longrightarrow row_1 = (a_o \times (a_1 + 1) \times 4$

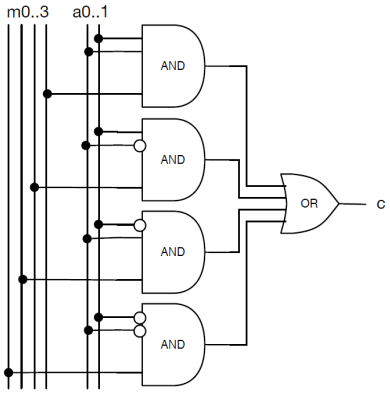
$row_0 = 1 \wedge 0 \wedge 1 = 0 \longrightarrow row_0 = (\underbrace{a_o + 1}_{\neg a_0}) \times (\underbrace{a_1 + 1}_{\neg a_1}) \times 5$

Calculate Circut in homorphic space

$c = 0 \vee 0 \vee 1 \vee 0 = 1$

$c = row_0 \circ row_1 \circ row_2 \circ row_3$

- Map to integer arithmetic



m = {1, 0, 1, 0} a = 01 m = {5, 4, 9, 6} a = {8, 3}

$$row_3 = 0 \wedge 1 \wedge 0 = 0 \longrightarrow row_3 = (a_o \times a_1 \times 6)$$

$$row_2 = 1 \wedge 1 \wedge 1 = 1 \longrightarrow row_2 = (a_o + 1) \times a_1 \times 9$$

$$row_1 = 0 \wedge 0 \wedge 0 = 0 \longrightarrow row_1 = (a_o \times (a_1 + 1) \times 4$$

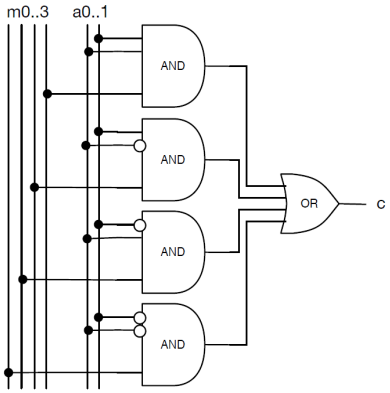
$$row_0 = 1 \wedge 0 \wedge 1 = 0 \longrightarrow row_0 = (\underbrace{a_o + 1}_{\neg a_0}) \times (\underbrace{a_1 + 1}_{\neg a_1}) \times 5$$

Calculate Circut in homorphic space

$$c = 0 \vee 0 \vee 1 \vee 0 = 1$$

$$c = row_0 \circ row_1 \circ row_2 \circ row_3$$

- Map to integer arithmetic



m = {1, 0, 1, 0} a = 01 m = {5, 4, 9, 6} a = {8, 3}

$$row_3 = 0 \wedge 1 \wedge 0 = 0 \longrightarrow row_3 = (8 \times 3 \times 6)$$

$$row_2 = 1 \wedge 1 \wedge 1 = 1 \longrightarrow row_2 = (8 + 1) \times 3 \times 9$$

$$row_1 = 0 \wedge 0 \wedge 0 = 0 \longrightarrow row_1 = (8 \times (3 + 1) \times 4$$

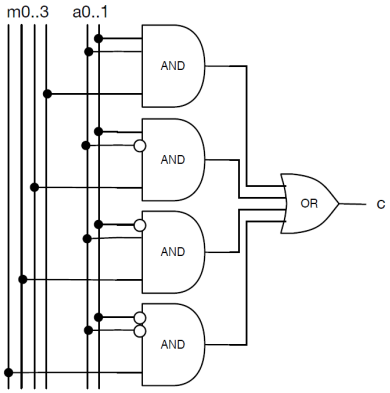
$$row_0 = 1 \wedge 0 \wedge 1 = 0 \longrightarrow row_0 = (\underbrace{8 + 1}_{\neg a_0}) \times (\underbrace{3 + 1}_{\neg a_1}) \times 5$$

Calculate Circut in homorphic space

$$c = 0 \vee 0 \vee 1 \vee 0 = 1$$

$$c = row_0 \circ row_1 \circ row_2 \circ row_3$$

- Map to integer arithmetic



m = {1, 0, 1, 0} a = 01 m = {5, 4, 9, 6} a = {8, 3}

$row_3 = 0 \wedge 1 \wedge 0 = 0 \longrightarrow row_3 = (8 \times 3 \times 6) = 144$

$row_2 = 1 \wedge 1 \wedge 1 = 1 \longrightarrow row_2 = (8 + 1) \times 3 \times 9 = 243$

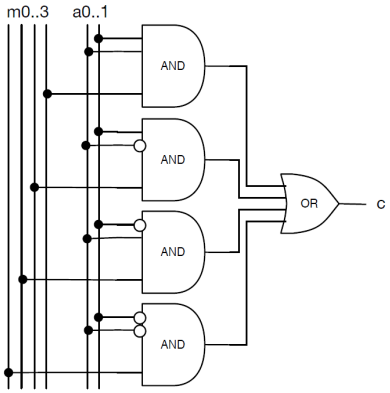
$row_1 = 0 \wedge 0 \wedge 0 = 0 \longrightarrow row_1 = (8 \times (3 + 1) \times 4 = 128$

$row_0 = 1 \wedge 0 \wedge 1 = 0 \longrightarrow row_0 = (\underbrace{8 + 1}_{\neg a_0}) \times (\underbrace{3 + 1}_{\neg a_1}) \times 5 = 180$

Calculate Circut in homorphic space

$c = 0 \vee 0 \vee 1 \vee 0 = 1$ $c = row_0 \circ row_1 \circ row_2 \circ row_3$

- Map to integer arithmetic



m = {1, 0, 1, 0} a = 01 m = {5, 4, 9, 6} a = {8, 3}

$$row_3 = 0 \wedge 1 \wedge 0 = 0 \longrightarrow row_3 = (8 \times 3 \times 6) = 144$$

$$row_2 = 1 \wedge 1 \wedge 1 = 1 \longrightarrow row_2 = (8 + 1) \times 3 \times 9 = 243$$

$$row_1 = 0 \wedge 0 \wedge 0 = 0 \longrightarrow row_1 = (8 \times (3 + 1) \times 4 = 128$$

$$row_0 = 1 \wedge 0 \wedge 1 = 0 \longrightarrow row_0 = (\underbrace{8 + 1}_{\neg a_0}) \times (\underbrace{3 + 1}_{\neg a_1}) \times 5 = 180$$

Calculate Circut in homorphic space

$$c = 0 \vee 0 \vee 1 \vee 0 = 1$$

$$c = 180 \circ 128 \circ 243 \circ 144$$
$$= 826087619 \mathrel{\hat{=}} 1$$

Fully homomorphic encryption

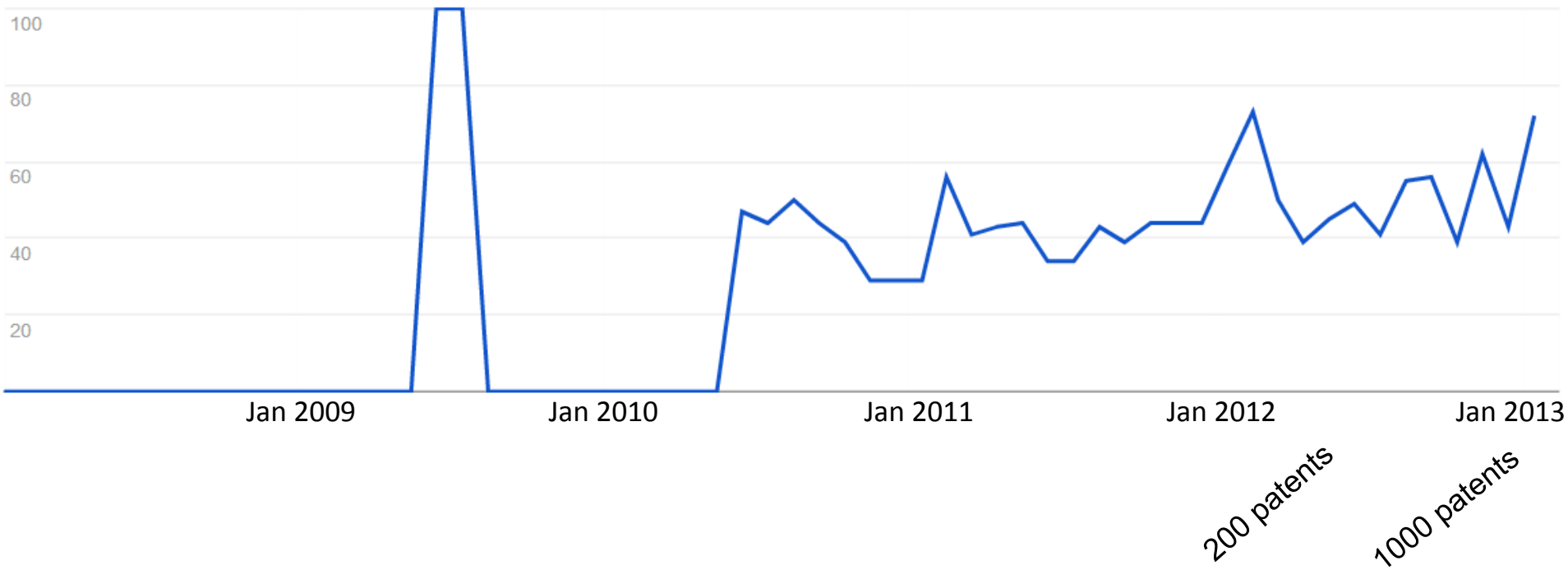
- "Holy Grail" of cryptography
- First proposed within a year of RSA development
  - 1979
  - Idea due to weird homomorphic property of RSA

(remember the area solver example)

- for more than 30 years:
  unclear whether FHE even possible
  - During that time: best one = Boneh-Goh-Nissim

(the one where only one multiplication was possible)

Google trends
"fully homomorphic encryption"



Gentry's approach

- **2009:** Craig Gentry shows fully homomorphic encryption in his doctoral thesis

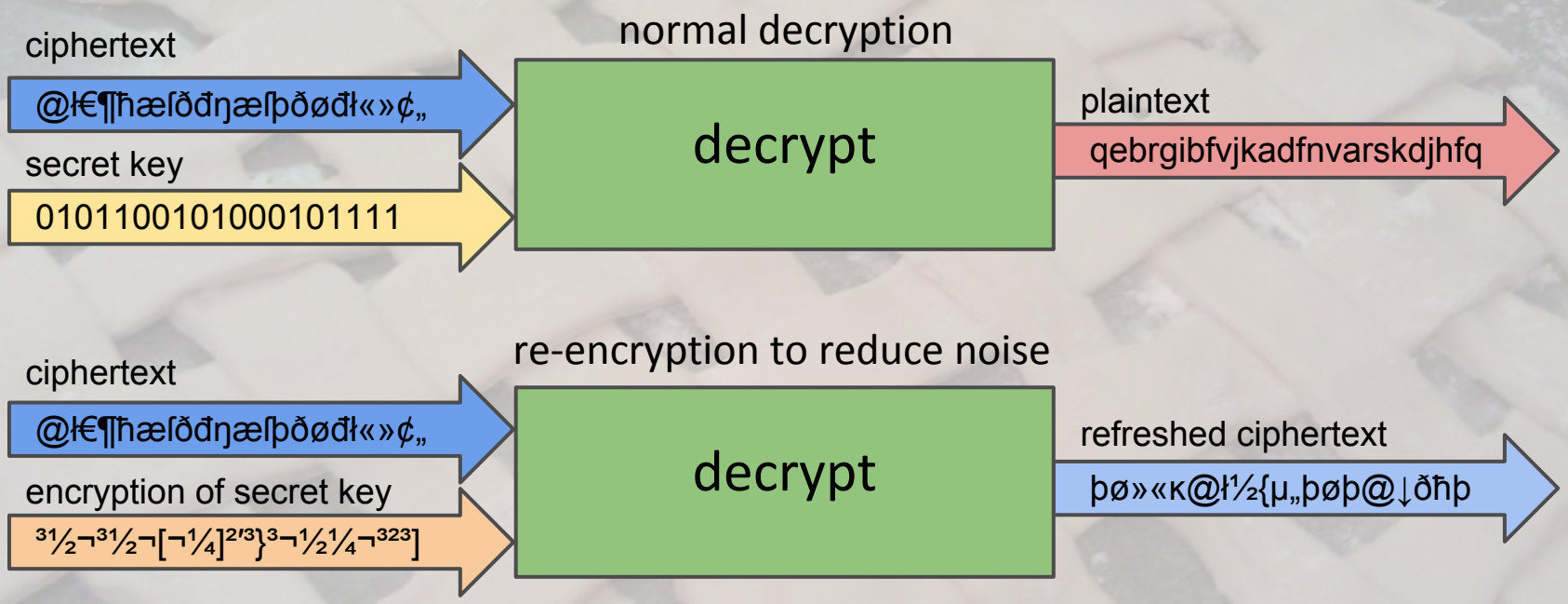
- Employs *somewhat homomorphic encryption* scheme using *ideal lattices*

  (based on "shortest lattice vector" problem used in cryptography, which is NP-hard)

- Scheme is *bootstrappable*
  - can evaluate its own decryption circuit

- Through recursive self-embedding, leads to FHE
  - ciphertexts are reencrypted, eliminating noise

Gentry's approach



Performance: SWHE

Bar-Ilan University
Dept. of Computer Science

Dimension	KeyGen	Enc amortized	Mult / Dec	degree
2048 800,000-bit integers	1.25 sec	.060 sec	.023 sec	~200
8192 3,200,000-bit integers	10 sec	.7 sec	.12 sec	~200
32768 13,000,000-bit integers	95 sec	5.3 sec	.6 sec	~200

Craig Gentry

https://www.youtube.com/watch?v=Y1TxCiOuoYY

Issues

1 000 000 000 000x

[...] a simple string search using homomorphic encryption is about a **trillion** times slower than without encryption. [1]

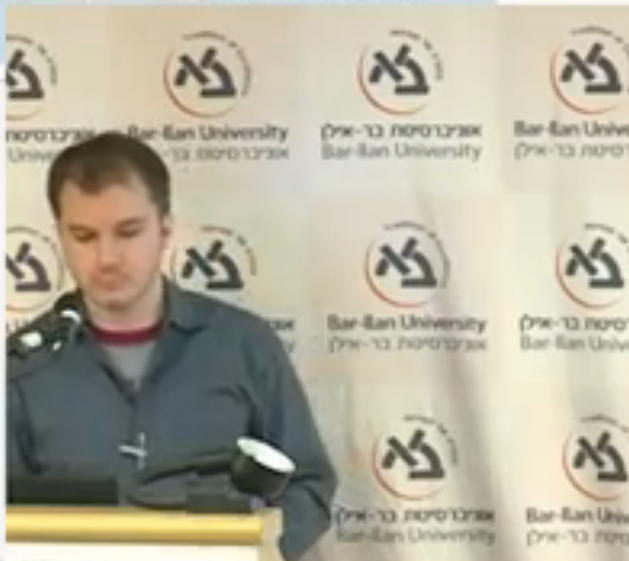
[1] CryptDB: A practical encrypted relational DBMS, RA Popa, N Zeldovich, H Balakrishnan, 2011

Performance: FHE

Noise reduction

Bar-Ilan University
Dept. of Computer Science

Dimension	KeyGen	PK size	ReCrypt
2048	40 sec	70 MByte	31 sec
8192	8 min	285 MByte	3 min
32768	2 hours	2.3 GByte	30 minute



Fully hom. enc. IRL

- HELib by Shai Halevi (2013)
  - Implementation of Brakerski-Gentry-Vaikuntanathan[1] scheme
  - Using many tricks / optimizations in literature[2][3] for speed
  - Does not implement bootstrapping (yet)

Performance

Modulus	Time for addition (ms)	Time for multiplication (ms)
257	0.7	39
8209	0.7	38
65537	2.9	177

{ Even numbers < 65537, 80 Bits of security }

[1] Zvika Brakerski, Craig Gentry, Vinod Vaikuntanathan: (Leveled) fully homomorphic encryption without bootstrapping. ITCS 201
[2] Nigel P. Smart, Frederik Vercauteren: Fully Homomorphic SIMD Operations. IACR Cryptology ePrint Archive 2011: 133 (2011)
[3] Craig Gentry and Shai Halevi and Nigel P. Smart Homomorphic Evaluation of the AES Circuit, CRYPTO 2012

Criticism

> Visions of a fully homomorphic cryptosystem have been dancing in cryptographers' heads for thirty years. [...] It will be years before a sufficient number of cryptographers examine the algorithm that we can have any confidence that the scheme is secure.

—Bruce Schneier, cryptographer, April 2013

Last few years

2009 First FHE by Gentry

Gentry & van Dijk simplify: no more lattices, integers instead

2010 Smart & Vercauteren reduce key size

2011 Gentry & Halevi present working FHE implementation

(the one w/ 2.3GB pub. key, runs on workstation)

Coron, Naccache & Tibouchi reduce public key size

Gentry & Brakerski show removal bootstrapping (Leveled FHE)

Smart & Vercauteren implement SIMD

2012 Gentry, Smart & Halevi elevate SIMD to general circuits

Boneh, Gentry & Halevi show another PoC using SWHE

Brakerski, Gentry & Vaikuntanathan improve Leveled FHE

Conclusion

- Implementations of HE that already work
  - HELib by Halevi (github.com/shaih/HElib)
    - GPL licensed
  - CryptDB by MIT (css.csail.mit.edu/cryptdb/)
- Privacy (in the cloud) could benefit greatly.
- When FHE ever gets fast, it could be The Next Big Thing™
  - Combining commercial usage of data with privacy

Conclusion

> *Should be usable in niche applications within a year or two.*
>
> — Halevi, 2012

Thanks for listening

Questions?

(12) **United States Patent**
Kerschbaum et al.

(10) **Patent No.: US 7,995,750 B2**
(45) **Date of Patent: Aug. 9, 2011**

(54) **PRIVACY-PRESERVING CONCATENATION OF STRINGS**

(57) **ABSTRACT**

A system for contributing to a concatenation of a first string and a second string may include a communication unit to receive an encrypted representation of a second share of the second string, the second string being identical to the second share of the second string combined with a first share of the second string and to send a rearranged representation of the encrypted representation of the second share of the second string to a second system. The system may further include a processing unit to rearrange a representation of the encrypted representation of the second share of the second string using a length value of a first share of the first string, the first string being identical to the first share of the first string combined with a second share of the first string.